خوارزمية بريم (Prim Algorithm)

تعتبر خوارزمية Prim من الخوارزميات الشرهة Greedy، هدفها إيجاد أصغر شجرة ممتدة (متصلة) Minimum Spanning Tree للبيان التي تطبق عليه، أي أنها:

- تحول البيان إلى شجرة لا تحوي حلقات،
- ويكون مجموع أوزان الأضلاع في هذه الشجرة الناتجة أصغر ما يمكن.

مبدأ عمل الخوارزمية:

1- انشئ مصفوفة mstSet وظيفتها تحديد العقد التي تم إضافتها لشجرة MST، وذلك بجعل قيمة العقدة true.

2- أدخل قيمة كبيرة جداً (غير منتهية مثلاً) لكل قيم المفاتيح في المصفوفة key. ثم أسند القيمة 0 للعقدة الأولى التي تم التقاطها أولاً.

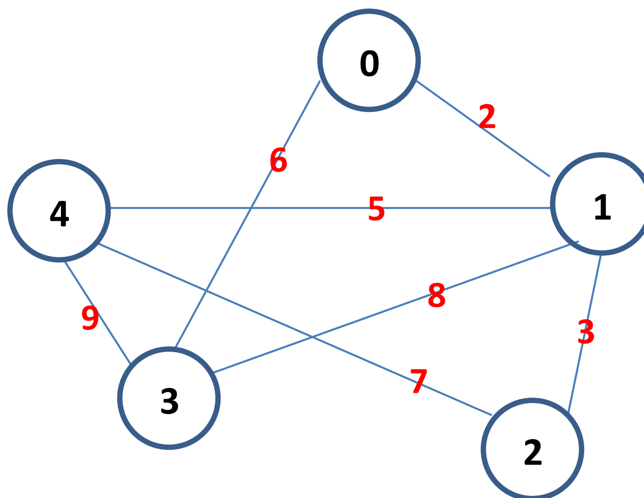
3- طالما أن هنالك عقد لم تضف إلى MST، أي أن المصفوفة mstSet مازالت تحوي false بقيمها، فقم بالتالي:

A. التقط العقدة u والتي مازالت غير موجودة في mstSet كما ولها أصغر قيمة مفتاح key،

B. ضم العقدة u إلى mstSet

C. قم بتحديث قيمة المفتاح لكل العقد المجاورة لـ u، يتم ذلك كالتالي: كل عقدة جوار لـ u هي v، إذا كان وزن الرابط u-v أقل من قيمة المفتاح السابق لـ v، فعندها حدث قيمة المفتاح ليصبح الوزن u-v.

مثال: اكتب كود خوارزمية Prim للبيان التالي بلغة C++، لتستطيع ايجاد شجرة MST علماً أن عقدة بداية MST هي العقدة 0.



ملاحظة: تحوي التعليمات على القيمة INT_MAX وهي تعبير محجوز في لغة C++ يمثل أكبر عدد صحيح، وفي خوارزميتنا يكافئ اللانهاية ∞ .

الحل:

```
// A C++ program for Prim's Minimum Spanning Tree (MST) algorithm. The
// program is for adjacency matrix representation of the graph

#include <iostream>
using namespace std;

// Number of vertices in the graph
#define V 5

// A utility function to find the vertex with minimum key value, from the
// set of vertices not yet included in MST
int minKey(int key[], bool mstSet[])
{
	// Initialize min value
	int min = INT_MAX, min_index;

	for (int v = 0; v < V; v++)
		if (mstSet[v] == false && key[v] < min)
			min = key[v], min_index = v;

	return min_index;
}

// A utility function to print the constructed MST stored in parent[]
void printMST(int parent[], int key[])
{
	cout << "Edge \tWeight\n";
	for (int i = 1; i < V; i++)
		cout << parent[i] << " - " << i << " \t"
			<< key[i] << " \n";
}

// Function to construct and print MST for a graph represented using
// adjacency matrix representation
void primMST(int graph[V][V])
{
	// Array to store constructed MST
	int parent[V];

	// Key values used to pick minimum weight edge in cut
	int key[V];

	// To represent set of vertices included in MST
	bool mstSet[V];
```

```
	// Initialize all keys as INFINITE
	for (int i = 0; i < V; i++)
		key[i] = INT_MAX, mstSet[i] = false;

	// Always include first 1st vertex in MST. Make key 0 so that this
	//vertex is picked as first vertex.
	key[0] = 0;

	// First node is always root of MST
	parent[0] = -1;

	// The MST will have V vertices
	for (int count = 0; count < V - 1; count++) {

// Pick the minimum key vertex from the set of vertices not yet
//included in MST
		int u = minKey(key, mstSet);

		// Add the picked vertex to the MST Set
		mstSet[u] = true;

		// Update key value and parent index of
		// the adjacent vertices of the picked vertex.
		// Consider only those vertices which are not
		// yet included in MST
		for (int v = 0; v < V; v++)

			// graph[u][v] is non zero only for adjacent
			// vertices of m mstSet[v] is false for vertices
			// not yet included in MST Update the key only
			// if graph[u][v] is smaller than key[v]
			if (graph[u][v]!=0 && mstSet[v] == false
				&& graph[u][v] < key[v])
				parent[v] = u, key[v] = graph[u][v];
	}

	// Print the constructed MST
	printMST(parent, key);
}

// Driver's code
int main()
{
	int graph[V][V] = { { 0, 2, 0, 6, 0 },
						{ 2, 0, 3, 8, 5 },
						{ 0, 3, 0, 0, 7 },
						{ 6, 8, 0, 0, 9 },
						{ 0, 5, 7, 9, 0 } };
```

```
	// Print the solution
	primMST(graph);
system("pause");
	return 0;
}
```

الخرج هو

Edge	Weight
0 - 1	2
1 - 2	3
0 - 3	6
1 - 4	5